POWER PROGRAMMING

The Windows Common Dialog Functions for Printing

BY RAY DUNCAN

Our tour of the Windows 3.1 common dialogs finally brings us to the subject of printing and printer setup. You may recall that during the discussion of the ChooseFont common dialog, I characterized traditional font management under Windows as programming voodoo, which I define as reliance on poorly documented procedures that are invoked ritualistically and are prone to failure for reasons that are only dimly understood. Well, the intricacies of Windows font management are only a foretaste of the surprises that lay in wait for you in the area of printing.

In theory, Windows applications are device-independent, and "painting" on the printer should be little different from painting on the screen. In reality, there are many considerations peculiar to use of the printer, including (but not limited to) the fact that the printer often supports different fonts than the screen supports; the printer may have higher or lower resolution than the screen; the printer may not contain enough memory to support all the individual pixels on a page; the printer can run out of paper or be taken off-line; the system may support several different printers at once; and two or more applications may try to send output to the printer simultaneously.

A Microsoft technical writer took note of this situation by starting the chapter on printing in the Windows 3.0 SDK *Guide to Programming* with a little gallows humor: "In Microsoft Windows, your application need not provide any printer-specific code; it can simply print to the current printer." The author then proceeded to describe, for 21 pages, exactly how to write the printer-specific code that your application didn't need to provide. Of course, we could assume that Microsoft was referring to differences between brands of printers, rather than between printers and other types of output devices—but even that doesn't make the statement true. Windows printer drivers are allowed quite a bit of latitude in the services they export, and a sophisticated graphical application cannot hope to generate high-quality hard copy unless it meticulously adapts its behavior to the capabilities of the printer driver.

The addition of a common printer dialog to Windows 3.1 relieves applications programmers of the need to design and test elaborate printing and setup dialogs.

BASICS OF WINDOWS PRINTING In Windows 3.0 and earlier, the issue of printing is complicated by the rather involved initialization an application needs to go through first. The application must call the Windows API function GetProfileString() to obtain the device= line describing the current printer from WIN.INI (or open and read WIN.INI directly), then parse the line to extract the printer brand and model, the printer type, and the printer output port. Once the printer is identified, the program must create a printer device context analogous to but not completely symmetric with a display device context, and present an elaborate dialog to the user allowing him to select the range of pages, the number of copies, the paper tray, the paper dimensions, and so forth. The application must also support a setup dialog that allows the user to reconfigure the printer or switch from one printer to another (in systems that support multiple printers).

With the device context in hand, the application can proceed to the actual business of printing. In Windows 3.0, the entire print job had to be framed with Escape() calls using the STARTDOC and ENDDOC parameters, allowing Windows to serialize printer output and keep the pages of one application from being interleaved with those of another. (The Escape() function is roughly equivalent to the DOS IOCTL function; it provides a direct channel of communication between the application and the printer driver.) GDI calls such as TextOut(), DrawText(), Rectangle(), and the like can be used with the printer DC in the expected fashion, but the application needs to keep track of page position and issue Escape() calls with the NEWFRAME parameter to eject pages when appropriate. If the output is mostly graphics, the application may have to segment the page into "bands" that won't overflow the printer's memory.

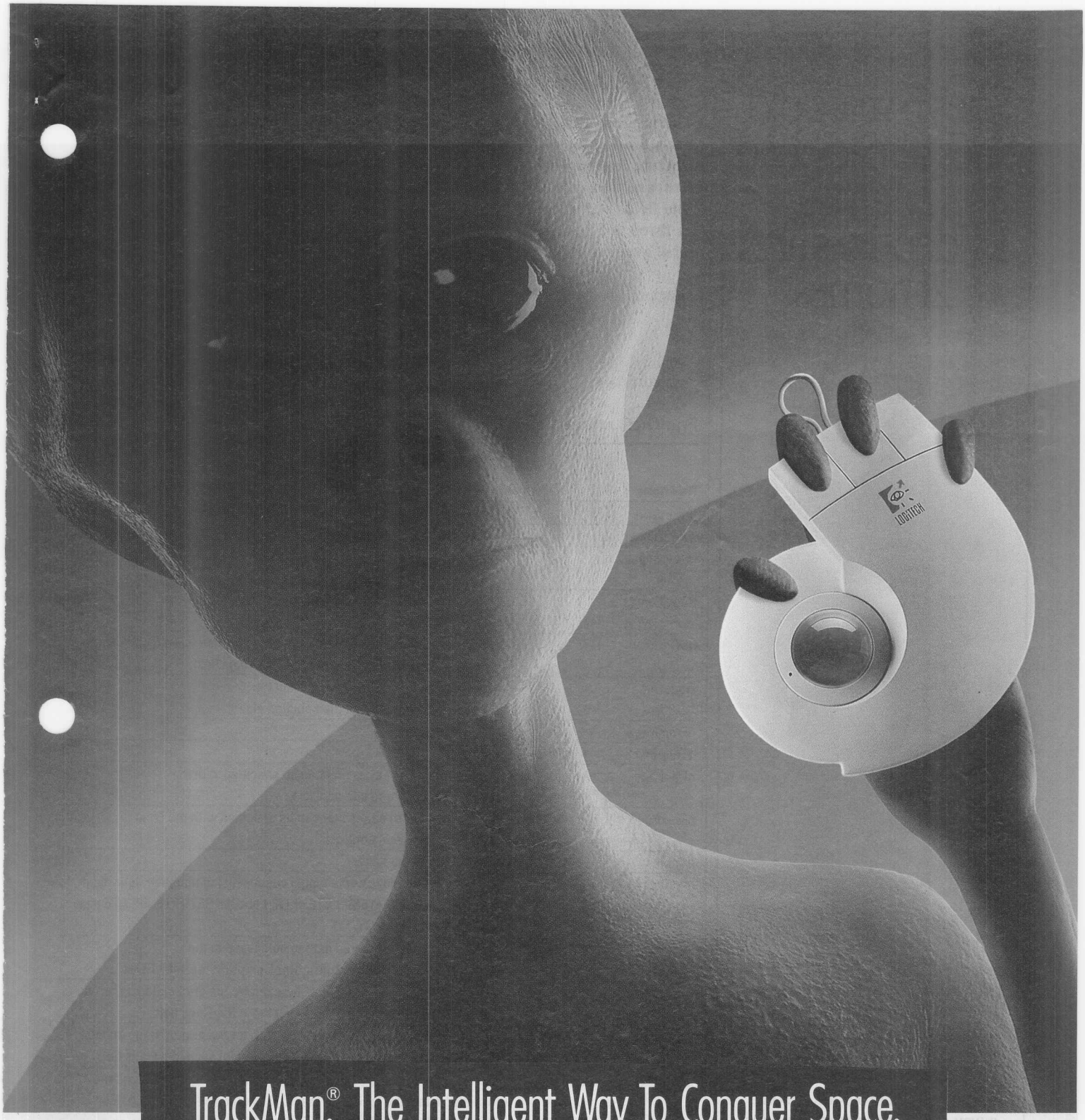
A Windows 3.0 application needs to be able to recover gracefully in case of a printer error, and must let the user abort the printing process at any time. This means that the application must contain a special callback procedure to be entered only in the event of a printer malfunction; this procedure is expected to put up an alert dialog that describes the problem to the user and lets him take action. The well-behaved Windows application should also display a free-floating dialog with a Cancel button during the entire process of printer output; fancier

apps audit the name of the file being printed, the current page number, and other useful information as well.

What changes does Windows 3.1 bring to this scheme? In general, the latest version of Windows drastically simplifies the application grunt work involved with printing. For typical applications, the catchall Escape() API function is superseded by the new and simpler API functions StartDoc(), StartPage(), EndPage(), EndDoc(), SetAbortProc(), AbortDoc(), and SpoolFile(). (The Escape() function is still available for backward compatibility and for applications that need to query printer capabilities.) The ChooseFont common dialog has options that let the application easily determine which fonts are supported for the printer, which for the screen, and which for both, while the addition of TrueType means that the application can leave behind most concerns about the availability (on the printer's end) of unusual font sizes or styles. Best of all, the addition of a common printer dialog relieves the applications programmer of the need to design and test elaborate printing and setup dialogs and their supporting callback functions.

You may well ask, isn't Microsoft's spiffing up of printer support in Windows 3.1 a case of bolting the barn door after the horse is long gone? It's true that few Windows programmers are inclined to revise and potentially break application source code that is already working, even if the revisions tend to simplify the code and improve maintainability. On the other hand, many programs will need to be thoroughly overhauled for OLE (object linking and embedding) and developers may use that opportunity to incorporate support for other Windows 3.1 enhancements, such as the common dialogs, TrueType, and multimedia. Furthermore, the new Windows 3.1 common dialogs and printing functions are exactly symmetric with the common dialogs and printing functions of NT/WIN32, so any conversion efforts now will be repaid in portability later.

USING PRINTDLG() The common printer dialog, whose code and resources are contained (like the other common dialogs) in the dynamic link library COMMDLG.DLL, is actually two dialogs in one. The COMMDLG.DLL entry



TrackMan.® The Intelligent Way To Conquer Space.

rackMan.
For Intelligent Beings,
Out of Space.

If your desk looks like it was hit by a meteorite, it's time to boldly go where thousands have gone before: to your local computer dealer for a TrackMan Stationary Mouse–the mouse that occupies a minimum of space. TrackMan's ergonomics are inspired, its compatibility guaranteed, and the three programmable buttons are a stroke of Windows™– crunching genius. All from the only company in the known universe to make over 15 million mice before making yours. 1-800-231-7717.

LOGITECH

point is called PrintDlg(), and you call it with a far pointer to a PRINTDLG data structure that controls various dialog options and receives the dialog's results. (The fields of the PRINTDLG structure are listed in Figure 1.) Here is the skeleton of a typical call to PrintDlg():

```
PRINTDLG pd;
.
.
.
PrintDlg(&pd);
```

PrintDlg() returns TRUE if the user successfully completes the dialog and clicks the OK button; it returns FALSE if the user clicks the CANCEL button or closes the dialog using the system menu, or if an error occurs. You can determine the cause of the error by calling CommDlgExtendedError() to get the error code.

The PRINTDLG structure contains numerous fields, but many of these can be ignored for routine applications programming. The crucial items to take note of in the PRINTDLG structure are the PD_PRINTSETUP and PD_RETURNDC bits in the Flags field. If the application calls PrintDlg() with the

The PRINTDLG Structure

Field Name	Description
StructSize	Specifies the length of the structure in bytes.
hwndOwner	Handle for the window that owns the dialog box. May be 0.
hDevMode	Handle to a memory block allocated by the common dialog function to hold a DEVMODE structure, if any.
hDevNames	Handle to a memory block allocated by the common dialog function to hold a DEVNAMES structure, if any.
hDC	Device context or information context for the printer, if requested by the PD_RETURNDC or PD_RETURNIC bit in the Flags field.
Flags	Contains various option flags that affect the initialization and behavior of the common dialog.
nFromPage	First page to print.
nToPage	Last page to print.
nMinPage	Minimum number of pages that can be specified with the From Page and To Page edit controls.
nMaxPage	Maximum number of pages that can be specified with the From Page and To Page edit controls.
nCopies	Number of copies to print.
hInstance	Identifies the data block containing a custom dialog template. Ignored if the PD_ENABLEPRINTTEMPLATE or FR_ENABLE SETUPTEMPLATE bit is not set in the Flags field.
lCustData	Specifies application-defined data to be passed to the function designated by the lpfnPrintHook or lpfnSetupHook fields.
lpfnPrintHook	Address of the application's "hook" function for the print dialog. Ignored if the PD_ENABLEPRINTHOOK bit is not set in the Flags field.
lpfnSetupHook	Address of the application's "hook" function for the printer setup dialog. Ignored if the PD_ENABLE SETUPHOOK bit is not set in the Flags field.
lpPrintTemplateName	Points to a string that specifies the name of the resource for a custom print dialog template. Ignored if the PD_ENABLE PRINTTEMPLATE bit is not set in the Flags field.
lpSetupTemplateName	Points to a string that specifies the name of the resource for a custom printer setup dialog template. Ignored if the PD_ENABLE SETUPTEMPLATE bit is not set in the Flags field.
hPrintTemplate	Handle for a memory block containing a custom print dialog template. Ignored if the PD_ENABLE PRINTTEMPLATEHANDLE bit is not set in the Flags field.
hSetupTemplate	Handle for a memory block containing a custom printer setup dialog template. Ignored if the PD_ENABLE SETUPTEMPLATEHANDLE bit is not set in the Flags field.

Figure 1: **Fields of the PRINTDLG data structure used by the PrintDlg() common dialog.**

BEST DOS.

BEST VALUE.

WINDOWS 3.1 COMPATIBLE

REPORT CARD
NOVEMBER 4, 1991
INFO WORLD

	DR DOS Version 6.0	MS-DOS Version 5.0
Performance		
Memory handling	Excellent	Very Good
User interface	Very Good	Very Good
Speed	Good	Very Good
Compatibility	Very Good	Very Good
Documentation	Excellent	Very Good
Setup	Good	Very Good
Ease of learning	Very Good	Very Good
Ease of use	Very Good	Good
Error handling	Satisfactory	Satisfactory
Support		
Support policies	Good	Satisfactory
Technical support	Satisfactory	Satisfactory
Value	Excellent	Excellent
Final Scores	7.6	7.1

WINNER BEST OF THE INTERFACE GROUP COMDEX/Fall '91 BYTE

BUYERS ASSURANCE SEAL INFO WORLD

1991 BYTE AWARD OF EXCELLENCE

DR DOS 6.0

1
Full DOS and Windows compatibility

2
*Nearly $400 worth of free utilities**

3
Double hard disk capacity

4
60-day money-back guarantee

*Superior memory management, optional "on-the-fly" file compression, high-performance disk caching, instant task switching, comprehensive password security, PC-to-PC transfer software, and more.

The critics agree: DR DOS® 6.0 delivers more of what today's PC users are looking for in a state-of-the-art DOS operating system:

"...tops [MS-DOS 5] with better disk performance, faster task switching, access to more RAM, and improved security, while providing full Windows™ support."
PC World, January 1992

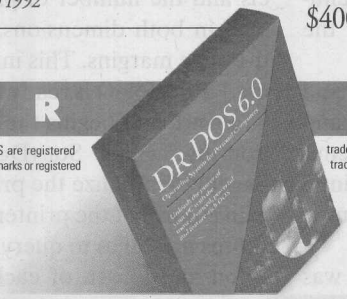
Flawless operation of all PC applications, and advanced capabilities for highest productivity:

"...complete MS-DOS 5 compatibility with more features and functions."
PC Magazine, November 12, 1991

And a complete package of indispensable utilities that normally would cost nearly $400 extra:

"...an excellent value." *InfoWorld, November 4, 1991*

Find out today why DR DOS 6.0 is the best DOS. And the best value, too. Get the facts by fax: 1-800-955-DOS6. Request document 703. Or call us today for details and the name of the DR DOS reseller nearest you: 1-800-274-4DRI.



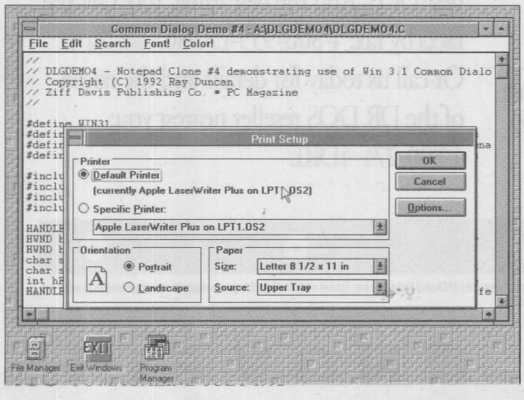
D R D O S 6 . 0

NOVELL

PD_PRINTSETUP flag, the function displays a Printer Setup dialog that allows the user to configure the printer or switch from one printer to another, as shown in Figure 2. All of the necessary grubbing around within WIN.INI and querying of printer capabilities is buried within the common dialog library. Here is typical code for presenting the Printer Setup dialog:

```
PRINTDLG pd;
.
.
.
memset(&pd, 0, sizeof(PRINTDLG));
pd.lStructSize = sizeof(PRINTDLG);
pd.Flags = PD_PRINTSETUP;
PrintDlg(&pd);
```

If an application calls PrintDlg() with the PD_RETURNDC flag, the function will display a Print dialog that allows the user to enter the range of pages and number of copies (see Figure 3). A Setup button in the dialog lets the user detour to the Printer Setup dialog, if desired. The PrintDlg() function then determines the type of printer and allocates a printer device context. After PrintDlg() returns, the program can extract the device context, number of copies, starting and ending page numbers, and other options from the PRINTDLG data structure. The structure also contains handles for memory blocks that contain DEVMODE and DEVNAMES structures, which the application can lock down and inspect for detailed information about the printer(s). Basic code for displaying a Print dialog and then calling a routine to print a file using the returned device context might look like this:




```
PRINTDLG pd;
.
.
.
memset(&pd, 0,
 sizeof(PRINTDLG));
pd.lStructSize =
 sizeof(PRINTDLG);
pd.Flags = PD_RETURNDC;
if(PrintDlg(&pd))
{
 PrintFile(pd.hDC);
 if(pd.hDevMode)
     GlobalFree(pd.hDevMode);
 if(pd.hDevNames)
     GlobalFree(pd.hDevNames);
 DeleteDC(pd.hDC);
}
```

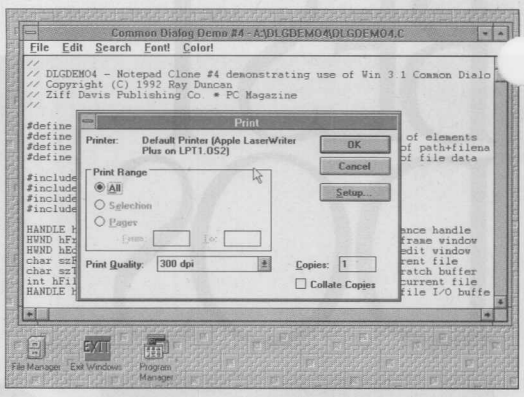


Figure 3: **This is a screen shot of DLGDEMO4, which shows the Print common dialog.**

Like the other common dialog functions, PrintDlg() has options that allow applications to filter all of the dialog's messages with a hook function or to replace the default dialog templates with custom templates. The needs of most applications, however, will be amply served by the default Print and Printer Setup dialogs and message handling.

THE DLGDEMO4 PROGRAM This issue's demonstration program, DLGDEMO4.C, is yet another version of the DLGDEMO programs printed in the last three issues. DLGDEMO4 is a clone of the Windows Notepad program that relies on a multiline edit control for primitive editing of a plain ASCII text file. Figure 4 contains extracts of the source code that are relevant to the printing capabilities of the program. The complete source code and executable program can be downloaded from PC MagNet.

Crucial changes in DLGDEMO4 are:

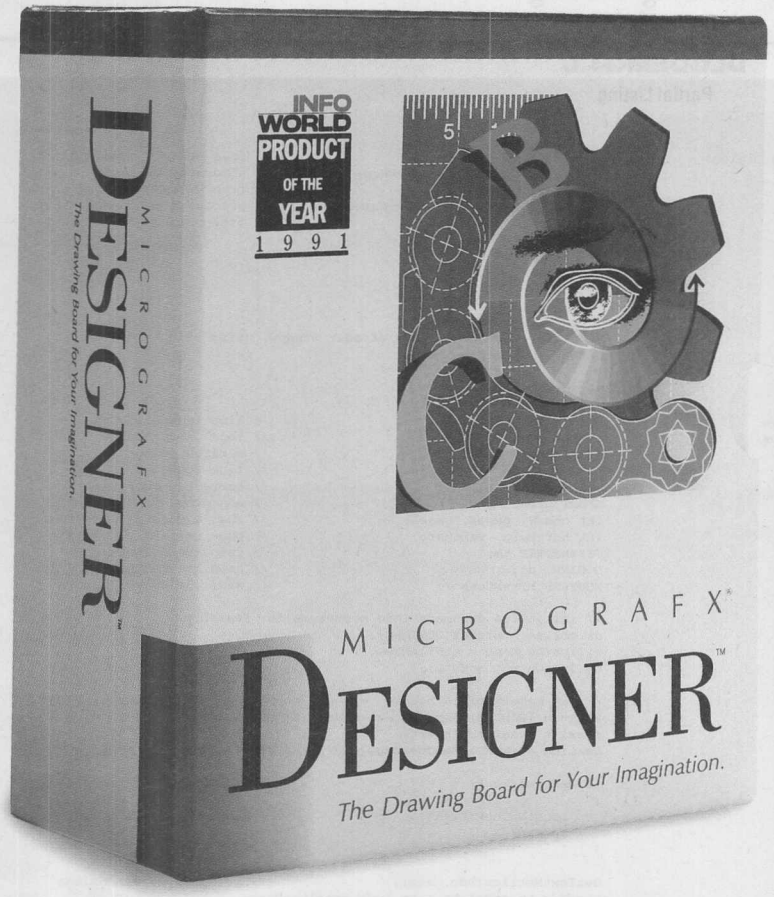
- the addition of Print and Print Setup options to the File pop-up menu in the resource script DLGDEMO4.RC;
- the addition of the routines DoMenuPrintSetup(), DoMenuPrint(), and PrintFile() to the source file; and
- the addition of the Print and Print Setup menu identifiers and the addresses of the DoMenuPrintSetup() and DoMenuPrint() routines to the table menuitems[] in DLGDEMO4.C

identifiers and the prototypes for the new routines.

Let's see what happens when a user requests a print operation. Windows processes the user's menu selection and sends a WM_COMMAND message with the IDM_PRINT identifier in wParam to DLGDEMO4's main callback routine, FrameWndProc(). FrameWndProc() decodes the message using the table messages[] and dispatches the message to the routine DoCommand(). DoCommand() decodes wParam using the table menuitems[] and dispatches the message to the routine DoMenuPrint().

When DoMenuPrint() gets control, it allocates a PRINTDLG data structure, initializes it, and calls PrintDlg() to display the Print common dialog. If PrintDlg() returns TRUE, DoMenuPrint() extracts the printer device context from the PRINTDLG data structure and passes it to the routine PrintFile(). PrintFile() locks down the memory block belonging to the edit control and determines the number of lines of text, exiting immediately if there's nothing to print. Otherwise, PrintDlg() calls GetTextMetrics() to get the vertical cell size (in pixels) of the current printer font, calls GetDeviceCaps() to get the page size in pixels and the number of pixels per logical inch in both dimensions, and calculates the page margins. This information gathered, PrintFile() is ready to get to work.

PrintFile() begins the printing operation by calling StartDoc() and StartPage() to initialize the printer driver and gain control of the printer output stream. It uses messages to query the starting po



PRODUCTS OF THE PAST.

PRODUCT OF THE YEAR.

The experts have cast their ballots. And once again, the winner is Designer!

Micrografx Designer™ is the *only* illustration software declared "1991 Product of the Year" by *InfoWorld*. The *only* illustration software named "Editor's Choice" by *PC Magazine* for three straight years. The *only* illustration software to rate five stars from *Software Digest*.

That's because no other software comes *near* the precision drawing power of Designer. With features like multiple layers. Dimensioning. Full-color, full-screen editing. And built-in type-handling software plus 180 Type 1 fonts worth over $5,000!

INFO WORLD PRODUCT OF THE YEAR 1991

"New text, blend and color features combine to keep Designer competitive in a fast-moving field. This precise, tool-rich drawing package has its roots in CAD, but it attracts other graphics users as well."

Features	Designer	Corel Draw
Drawing layers	64	1
Dimensioning	Yes	No
Object snap	Yes	No
Maximum drawing size	132" × 132"	17" × 17"
Edit in full color	Yes	No
Clip art images	Over 1,700	750
Type 1 fonts	180	0
PageMaker 4.0 filter	Yes	No
On line help screens	Yes	No
24-hour support	Yes	No

Move up to Designer now and get 70% off!

If you use Corel Draw or Arts & Letters, call us with your serial number and we'll upgrade you to Micrografx Designer for only $199 (regularly $695) plus $10 shipping and handling. If you're not *blown away* by how much more you can do with Micrografx Designer, return it for a full refund.* This is a limited-time offer, so act now!

Why settle for a product of the past when the future of PC illustration is Micrografx Designer? Call today to order or request a free working model!

30-day money-back guarantee!

1-800-998-0149

MICROGRAFX®

*Shipping & handling charge not refundable. Other restrictions may apply. Please call customer service for return authorization number. Micrografx reserves the right to cancel or amend the above offer at any time.

Designer system requirements: 286 (386 recommended) IBM PC or compatible, or PS/2. 1 MB RAM (2 MB RAM recommended). 20 MB (or larger) hard disk. Windows 3.0. DOS 3.1 (or higher). Mouse or digitizing pad. Windows-compatible monitor.

CIRCLE 361 ON READER SERVICE CARD

DLGDEMO4.C

Partial Listing

```
//
// DLGDEMO4 - Notepad Clone #4 demonstrating use of Win 3.1 Common Dialogs
// Copyright (C) 1992 Ray Duncan
// Ziff Davis Publishing Co. * PC Magazine
//

#define WIN31

#include "string.h"
#include "windows.h"
#include "commdlg.h"
#include "dlgdemo4.h"

    .
    .
    .

struct decodeWord {                         // structure associates
    WORD Code;                              // messages or menu IDs
    LONG (*Fxn)(HWND, WORD, WORD, LONG); }; // with a function

    .
    .
    .

//
// Table of menubar item IDs and their corresponding functions.
//
struct decodeWord menuitems[] = {
    IDM_NEW, DoMenuNew,
    IDM_OPEN, DoMenuOpen,
    IDM_SAVE, DoMenuSave,
    IDM_SAVEAS, DoMenuSaveAs,
    IDM_PSETUP, DoMenuPrintSetup,
    IDM_PRINT, DoMenuPrint,
    IDM_EXIT, DoMenuExit,
    IDM_UNDO, DoMenuUndo,
    IDM_CUT, DoMenuCut,
    IDM_COPY, DoMenuCopy,
    IDM_PASTE, DoMenuPaste,
    IDM_DELETE, DoMenuDelete,
    IDM_FIND, DoMenuFind,
    IDM_REPLACE, DoMenuReplace,
    IDM_FONT, DoMenuFont,
    IDM_COLOR, DoMenuColor, } ;

    .
    .
    .

//
// DoMenuPrintSetup -- process File-PrintSetup command from menu bar.
//
LONG DoMenuPrintSetup(HWND hWnd, WORD wMsg, WORD wParam, LONG lParam)
{
    PRINTDLG pd;                            // used by common dialog

    memset(&pd, 0, sizeof(PRINTDLG));       // zero out entire structure
    pd.lStructSize = sizeof(PRINTDLG);      // length of structure
    pd.hwndOwner = hWnd;                    // owner window
    pd.Flags = PD_PRINTSETUP;               // option flags

    PrintDlg(&pd);                          // display print setup
    return(0);                              // common dialog
}

//
// DoMenuPrint -- process File-Print command from menu bar.
//
LONG DoMenuPrint(HWND hWnd, WORD wMsg, WORD wParam, LONG lParam)
{
    PRINTDLG pd;                            // used by common dialog

    memset(&pd, 0, sizeof(PRINTDLG));       // zero out entire structure
    pd.lStructSize = sizeof(PRINTDLG);      // length of structure
    pd.hwndOwner = hWnd;                    // owner window
    pd.Flags = PD_RETURNDC | PD_NOPAGENUMS  // option flags
        | PD_NOSELECTION | PD_NOSELECTION | PD_HIDEPRINTTOFILE;

    if(PrintDlg(&pd))                       // display print common dialog
    {
        PrintFile(pd.hDC);                  // print text in edit control

        if(pd.hDevMode)                     // free DEVMODE memory block
            GlobalFree(pd.hDevMode);        // allocated by common dialog
        if(pd.hDevNames)                    // free DEVNAMES memory block
            GlobalFree(pd.hDevNames);       // allocated by common dialog
        DeleteDC(pd.hDC);                   // free printer device context
    }

    return(0);
}

//
// PrintFile - print contents of edit window, using printer device
// context supplied by caller.
//
VOID PrintFile(HDC hdc)
{
    WORD cLine, iLine;                      // line length, index
    HANDLE hText;                           // local heap handle
    PSTR pText;                             // scratch pointer
    WORD maxLine, curLine;                  // line counters
    WORD curY;                              // current Y coordinate
    HFONT hFont;                            // non-prop. font handle
    int charY, pageX, pageY;                // char & page Y dimensions
    int horzMarg, vertMarg;                 // page margins
    TEXTMETRIC tm;                          // info about font
    DOCINFO di;                             // used by StartDoc
    HCURSOR hPrevCursor;                    // save old cursor handle

    // initialize document info structure for StartDoc
    di.cbSize = sizeof(DOCINFO);
    di.lpszDocName = szFileName;
    di.lpszOutput = NULL;

    // get memory block handle, convert to pointer, get number of lines
    hText = (HANDLE) SendMessage(hEdit, EM_GETHANDLE, 0, 0L);
    pText = LocalLock(hText);
    maxLine = (WORD) SendMessage(hEdit, EM_GETLINECOUNT, 0, 0);

    if(maxLine == 0)                        // if there's nothing to
    {                                       // print, unlock memory
        LocalUnlock(hText);                 // block and bail out now
        return;
    }

    GetTextMetrics(hdc, &tm);               // get various dimensions
    charY = tm.tmHeight + tm.tmExternalLeading;
    pageY = GetDeviceCaps(hdc, VERTRES);
    horzMarg = GetDeviceCaps(hdc, LOGPIXELSX) / 2;
    curY = vertMarg = GetDeviceCaps(hdc, LOGPIXELSY) / 2;

    // get old cursor handle, change cursor to hourglass
    hPrevCursor = SetCursor(LoadCursor(hInst, IDC_WAIT));

    StartDoc(hdc, &di);                     // begin the print job
    StartPage(hdc);

    // print contents of edit control, line by line
    for(curLine = 0; curLine < maxLine; curLine++)
    {
        // get character index and length for current line
        iLine = (WORD) SendMessage(hEdit, EM_LINEINDEX, curLine, 0);
        cLine = (WORD) SendMessage(hEdit, EM_LINELENGTH, iLine, 0);

        // now draw the current line using printer device context
        TextOut(hdc, horzMarg, curY, &pText[iLine], cLine);

        curY += charY;                      // advance down page

        if(curY >= pageY - vertMarg)        // reached end of page?
        {
            curY = vertMarg;                // yes, reset Y coordinate
            EndPage(hdc);                   // and force new page
        }
    }

    EndPage(hdc);                           // send final new-page
    EndDoc(hdc);                            // and end the print job

    SetCursor(hPrevCursor);                 // restore old cursor shape
    LocalUnlock(hText);                     // unlock edit control
    return;                                 // memory block & return
}
```

Figure 4: **These are extracts from the source code for DLGDEMO4.C; they show the routines specific to the program's printing and printer setup functionality.**

time, and prints each line with a call to TextOut(). PrintFile() maintains a "current Y coordinate" for the page, incrementing this coordinate by the character cell's vertical size as each line is printed and resetting the coordinate and requesting a new page when the lower limit of the page is reached. After all lines have been printed, PrintFile() calls EndPage() and EndDoc() to eject the last page and terminate the print stream. In order to keep the source code simple, I haven't shown the code associated with tab expansion, font selection, or error handling, or with the Cancel dialog window.

THE IN-BOX Please send your questions, comments, and suggestions to me at any of these electronic-mail addresses:

PC MagNet: 72241,52

MCI Mail: rduncan

BIX: rduncan □